Seminar Metaheuristiken und Approximationsalgorithmen: Populationsbasierte Metaheuristiken: Grundlagen und Schwarmintelligenz

Michael Hamann

23. August 2012

Inhaltsverzeichnis

1 Einleitung

Bei populationsbasierten Metaheuristiken wird im Gegensatz zu Metaheuristiken mit einzelnen Lösungen nicht nur eine Lösung weiter verfolgt, sondern eine Menge von Lösungen, die Population. Hierdurch kann eine größere Anzahl an Lösungsansätzen für eine Probleminstanz gleichzeitig systematisch (oder zufällig) weiterentwickelt werden und Entwicklungen, die sich als gut herausstellen, können von anderen Individuen der Population übernommen werden.

Die einzelnen Verfahren sind meist durch in der Natur beobachtete Phänomene inspiriert wie der Fortpflanzung von Lebewesen im Falle der evolutionären Algorithmen, dem Verhalten von Ameisen für die Ameisenalgorithmen und Fisch- und Vogelschwärmen für Partikelschwarmoptimierung.

Am Anfang der verschiedenen Verfahren für populationsbasierte Metaheuristiken steht die Auswahl einer *initialen Population*. Die initiale Population kann dabei zufällig gewählt werden, es kann aber auch eine möglichst große Diversifikation der initialen Lösungen oder bereits eine erste heuristische Optimierung versucht werden, weitere Details zur Wahl von initialen Populationen werden in Abschnitt 2 erläutert.

In einem zweiten Schritt werden dann iterativ neue Lösungen generiert. Hierbei gibt es verschiedene Ansätze. Zum einen kann die Population an sich teilweise oder auch ganz ersetzt werden durch eine neue Population, die z.B. mit evolutionären Algorithmen generiert wird, hierauf soll in dieser Arbeit allerdings nicht näher eingegangen werden. In dieser Arbeit wird in Abschnitt 3 näher auf schwarmbasierte Verfahren eingegangen. Bei diesen wird die Population nicht ersetzt, sondern bewegt sich als Schwarm durch den Entscheidungsraum, sprich die Menge der möglichen Eingabeparameter für eine (gültige) Lösung, die von einer Zielfunktion bewertet wird. Optional kann bei schwarmbasierten Metaheuristiken zusätzlich zu den Informationen aus dem letzten Iterationsschritt auch noch ein zentrales Gedächtnis verwendet werden, dieses wird auch als *Blackboard* bezeichnet.

Am Ende des Verfahrens steht eine *Abbruchbedingung*. Diese kann eine zeitliche Limitierung sein (gemessen in Zeit oder auch Anzahl von Iterationen oder Auswertungen der Zielfunktion) oder ein adaptives Kriterium wie z.B. die Annäherung an eine bekannte Schranke. Zusätzlich können bei populationsbasierten Verfahren auch Kriterien wie die Diversität der Population als Abbruchbedingung herangezogen werden.

Die wesentliche Grundlage für diese Arbeit bilden die Kapitel 3.1 und 3.6 des Buches „Metaheuristics: From Design to Implementation“ von El-Ghazali Talbi [Tal09], wo

weitere Quellen verwendet wurden, werden diese erwähnt.

Zunächst wird in Abschnitt 2 näher erläutert, wie initiale Populationen allgemein generiert werden können und wie sich die verschiedenen Möglichkeiten auf den weiteren Verlauf der Verfahren auswirken. Danach wird näher auf schwarmbasierte Verfahren eingegangen, es werden Ameisenalgorithmen und Partikelschwarmoptimierung vorgestellt. Da die Konzepte hinter diesen Verfahren zwar bildlich meist recht gut vorstellbar sind, wenn man ein konkretes Optimierungsproblem betrachtet aber doch recht weit von der Anwendung entfernt scheinen, soll am Ende mit einem ausführlichen Anwendungsbeispiel an dem Problem der Knotenfärbung von Graphen exemplarisch gezeigt werden, wie sich sowohl Ameisenalgorithmen als auch Partikelschwarmoptimierung zur möglichst optimalen Lösung von Knotenfärbungsproblemen einsetzen lassen.

2 Auswahl der initialen Population

Bei populationsbasierten Metaheuristiken ist die initiale Population der Ausgangspunkt der Suche nach einer möglichst optimalen Lösung. Da die im iterativen Prozess gefundenen Lösungen auf den bislang gefundenen Lösungen basieren, ist es wichtig, dass die initiale Population zum einen selbst divers ist und zum anderen auch vielfältige Kombinationen erlaubt, sprich durch Kombination existierender Lösungen leicht eine gute Abdeckung des Entscheidungsraumes erreicht werden kann.

Bei den Verfahren zur Wahl der initialen Population kann im Wesentlichen zwischen drei verschiedenen Typen von Verfahren unterschieden werden: komplett zufällige Verfahren, Verfahren, die versuchen, die Diversität zu maximieren, und Verfahren, die bereits bei der Wahl der initialen Population die Lösung heuristisch optimieren.

2.1 Zufallsgeneratoren

In der Praxis sind „zufällig“ generierte Werte nie wirklich algorithmisch unabhängig und statistisch guter Zufall muss nicht immer auch eine gute Ausgangsbasis für populationsbasierte Metaheuristiken darstellen. Deshalb ist auch bei zufälligen initialen Populationen die Auswahl eines geeigneten Algorithmus' wichtig.

Bei Zufallsgeneratoren wird zwischen pseudo-zufälligen Zahlensequenzen und quasi-zufälligen Zahlensequenzen unterschieden. Im Gegensatz zu pseudo-zufälligen Zahlensequenzen wird bei Generatoren für quasi-zufälligen Zahlensequenzen auch die Streuung der Zahlen betrachtet, hierdurch wird im Allgemeinen eine höhere Diversität erreicht.

In der Praxis sind bei populationsbasierten Metaheuristiken pseudo-zufällige Zahlensequenzen am populärsten.

Zufällige Werte sind nicht unbedingt gleichmäßig über den Entscheidungsraum verteilt, sondern können auch sehr dicht beieinander liegen. Um dieses Problem zu lösen wurden Verfahren zur Diversifikation der initialen Population entwickelt:

2.2 Diversifikation

Um eine möglichst große Diversifikation zu erreichen, kann zum einen der Entscheidungsraum in Blöcke geteilt werden und in jedem Block ein zufälliger Vektor berechnet werden. Zum anderen kann eine Serie pseudo-zufälliger Lösungen verwendet werden, wobei allerdings alle Lösungen aussortiert werden, die zu dicht an bereits vorher gewählten Lösungen

sind, dies ist der Simple sequential inhibition process (SSI). Da hierbei zum einen immer die jeweiligen Abstände berechnet werden müssen und zum anderen zahlreiche Lösungen verworfen werden ist der SSI-Prozess deutlich aufwändiger in der Berechnung als die anderen Verfahren.

2.3 Heuristische Initialisierung

Eine Heuristik wie eine Greedy-Heuristik kann benutzt werden, um initiale Populationen zu generieren. Dabei kann zwar auch mit Zufall gearbeitet werden, im Allgemeinen wird dies aber zu einer deutlich weniger diversen initialen Population führen und damit zu einer raschen Konvergenz. Je nach konkretem Einsatzzweck kann dies gewünscht sein (z.B. bei sehr aufwändigen Berechnungen, bei denen eine rasche Konvergenz gewünscht und auch sinnvoll ist).

2.4 Einfluss der initialen Population

Maaranen et al. [MMP07] haben gezeigt, dass der Einfluss der initialen Population durchaus bedeutend sein kann und dann in Experimenten untersucht, wie sich verschiedene Algorithmen für die Wahl der initialen Population sowohl auf Zwischenergebnisse nach 10 und 20 Generationen als auch auf die Lösung am Ende auswirken.

Untersucht wurde der Einfluss initialer Populationen bei genetischen Algorithmen. Als eher akademisches Beispiel wurde dabei zunächst der Parameterbereich der initialen Lösung auf die oberen bzw. unteren 80% beschränkt. Hierdurch ergaben sich insbesondere am Anfang deutliche Unterschiede bezüglich der Qualität der Lösung, die aber teilweise auch lange bestehen blieben und sich damit auch auf die Konvergenz insgesamt ausgewirkt haben.

Die Experimente haben gezeigt, dass eine gute genetische Diversität (d.h. es sind vielfältigere Kombinationen der verschiedenen Parametervektoren möglich) wie sie z.B. bei den pseudo-Zufälligen Punkten vorliegt zu einer schnelleren Konvergenz führt, eine höhere Diversifikation mit niedrigerer genetischer Diversität wie sie bei dem SSI-Prozess erreicht wird dagegen zu einer langsameren Konvergenz mit einer besseren Qualität der endgültigen Lösung.

Insgesamt konnten allerdings keine signifikanten Unterschiede festgestellt werden, die eine klare Überlegenheit eines bestimmten Verfahrens gezeigt hätten. Deshalb ist die in der Praxis häufige Verwendung pseudo-zufälliger Punkte durchaus gerechtfertigt. Pseudozufällige Punkte können schnell generiert werden und führen zu einer schnellen Konvergenz, so dass bei komplexen Berechnungen auch ein vorzeitiger Abbruch des Verfahrens noch relativ gute Ergebnisse liefern kann.

3 Schwarmbasierte Verfahren

Lebewesen bewegen sich in der Natur häufig in Gruppen oder Schwärmen. Dies hat den Vorteil, dass einzelne Individuen nicht selbst alles wissen und entdecken müssen, sondern von dem Wissen und den Entdeckungen ihrer Artgenossen profitieren können. Dies kann sowohl bei der Suche nach Nahrung als auch bei der Entdeckung von Gefahren nützlich sein. Bei bestimmten Tieren wie z.B. Ameisen, Fischen oder Vögeln ist dieses Verhalten besonders ausgeprägt beobachtbar und hat die im Folgenden vorgestellten Verfahren inspiriert.

3.1 Ameisenalgorithmen

Ameisenalgorithmen sind ein Beispiel für eine populationsbasierte Metaheuristik, die einen zentralen Speicher verwendet. Dieser ist inspiriert von den Pheromonen, die Ameisen auf den von ihnen zurückgelegten Pfaden hinterlassen, um ihre Artgenossen z.B. den Weg zu Futterstellen zu zeigen. Dabei kehren Ameisen von näheren Futterstellen schneller zurück, können dadurch häufiger zur Futterstelle gehen und erzeugen daher stärkere Pheromonpfade, wodurch immer mehr Ameisen auf diese Pfade gelenkt werden. Dieser Effekt wird dadurch verstärkt, dass die Pheromonpfade mit der Zeit schwächer werden.

Bei Ameisenalgorithmen zeigt der Pheromonpfad gute Wege bzw. Komponenten von guten Lösungen an. Wie in der Natur werden die Pheromone dabei mit der Zeit schwächer. Die Pheromonpfade werden bei Ameisenalgorithmen als Pheromonmatrix dargestellt. Die Einträge in der Matrix entsprechen den Bewertungen der verschiedenen Komponenten von Lösungen, welche Komponenten in der Matrix dargestellt werden, hängt dabei von dem konkreten Problem ab.

Ameisenalgorithmen wurden ursprünglich zur Lösung von Routingproblemen wie dem Problem des Handlungsreisenden entworfen, werden mittlerweile aber auch zur Lösung anderer Probleme verwendet.

Zusätzlich zu der Pheromonmatrix wird bei Ameisenalgorithmen auch eine problemspezifische, randomisierte Greedy-Heuristik verwendet. Die Wahrscheinlichkeiten, bestimmte Optionen zu wählen, verändern sich dabei im Laufe der Zeit durch den Pheromonpfad.

Am Anfang konstruiert jede Ameise eine zufällige Lösung unter Verwendung der randomisierten Greedy-Heuristik. Danach werden iterativ abwechselnd die Pheromonmatrix aktualisiert und neue Lösungen unter Verwendung der von der Pheromonmatrix beein-

flussten Greedy Heuristik generiert. Das Verfahren wird beendet, wenn eine Abbruchbedingung erfüllt ist.

Die Aktualisierung der Pheromonmatrix erfolgt in zwei Schritten: Im ersten Schritt werden die Pheromone um einen festen Faktor verringert. Im zweiten Schritt werden dann basierend auf den neu gefundenen Lösungen bestimmte Pheromone verstärkt oder abgeschwächt. Bei der Verstärkung/Abschwächung der Pheromone gibt es verschiedene Strategien. Zum einen können die Ameisen die Pheromone unabhängig voneinander aktualisieren, entweder nach jedem Schritt oder nur wenn eine vollständige Lösung gefunden wurde. Zum anderen können die Aktualisierungen auch zentral koordiniert werden. Dabei können zum einen die besten k Lösungen entweder nach Qualität oder nach Rang gewichtet die Einträge in der Pheromonmatrix erhöhen, zum anderen können aber auch die Ameisen mit den schlechtesten Lösungen die Einträge in der Pheromonmatrix, die zu den Komponenten ihrer Lösungen gehören, abschwächen. Zusätzlich dazu kann auch noch ein Minimum (oder Maximum) der Pheromone definiert sein, so dass jede Option eine minimale bzw. maximale Wahrscheinlichkeit hat, ausgewählt zu werden, um eine zu einseitige Entwicklung der Lösungen zu verhindern.

Insgesamt gibt es also drei wesentliche Entscheidungen, die bei der Modellierung eines Problems als Ameisenalgorithmus getroffen werden müssen: Was stellen die Pheromone dar, wie werden Lösungen konstruiert und wie werden die Pheromone aktualisiert.

Ein wichtiger Faktor für den Erfolg eines Ameisenalgorithmus ist, wie die Bewertung der Qualität einer Lösung erfolgt, da dies entscheidet, wie die Pheromonpfade aktualisiert werden.

Da die Pheromonmatrix zunächst nur eine endliche Anzahl Werte enthält, ist das Modell der Ameisenalgorithmen nicht direkt auf kontinuierliche Probleme anwendbar. Es gibt aber Adaptierungen von Ameisenalgorithmen für kontinuierliche Probleme, hierbei wird z.B. die Pheromonmatrix durch eine kontinuierliche Dichtefunktion ersetzt.

Typische Anwendungen für Ameisenalgorithmen sind Probleme des Scheduling, Routenplanung sowie die Berechnung von Zuordnungen, hierbei auch jeweils insbesondere in dynamischen Szenarien.

3.2 Partikelschwarmoptimierung

Partikelbasierte Verfahren sind vom Schwarmverhalten von Fischen und Vögeln inspiriert und modellieren die Bewegungsenergie in Schwärmen.

Eine feste Menge von N Partikeln bewegt sich bei partikelbasierten Verfahren in einem D-dimensionalen Entscheidungsraum und hat eine bestimmte Richtung und Geschwindigkeit. Wohin sich die einzelnen Partikel bewegen hängt dabei sowohl von der besten selbst gefundenen Lösung als auch der besten Lösung in der Nachbarschaft eines Partikels ab.

Die Nachbarschaft eines Partikels ist durch eine Graphstruktur definiert. Die Nachbarschaft kann dabei sehr global sein, dies führt zu einer schnelleren Konvergenz oder auch eher lokal sein, dies führt zu mehr Diversität.

Konkret wird ein Partikel durch eine Menge von Vektoren definiert:

- x definiert die aktuelle Position des Partikels im Entscheidungsraum
- p definiert den Ort der bislang besten gefundenen Lösung des Partikels
- v ist der Richtungsvektor, er zeigt in die Richtung, in die sich der Partikel bewegt, sein Betrag ist die Geschwindigkeit, mit der sich der Partikel bewegt

Um zu definieren, welche Lösung die „beste“ Lösung ist, wird außerdem – wie auch schon bei den Ameisenalgorithmen – eine Bewertungsfunktion für Lösungen benötigt. Da diese entscheidet, welche Lösungsansätze weiter verfolgt werden, ist es wichtig, dass diese Funktion aussagekräftige Bewertungen trifft und insbesondere auch feine Unterschiede erkennen kann.

Ein Partikelschwarm kann auch als Zellularautomat aufgefasst werden, da der nächste Zustand eines Partikels ausschließlich von seinem eigenen vorherigen Zustand sowie dem Zustand seiner Nachbarn abhängt.

Seien $\rho_1 \in [0, C_1]$, $\rho_2 \in [0, C_2]$ zwei Zufallsvariablen, C_1 und C_2 zwei Faktoren, die die Beeinflussung der neuen Richtung eines Partikels durch seinen eigenen Erfolg bzw. den Erfolg seiner Nachbarn beschreiben, g_i die Position der besten Lösung der Nachbarn von p_i und w die Trägheit, die bestimmt, wie stark die alte Richtung die neue Richtung des Partikels beeinflusst. Dann ist die Richtung eines Partikels zum Zeitpunkt t durch die folgende Formel definiert:

$$v_i(t) := wv_i(t-1) + \rho_1(p_i - x_i(t-1)) + \rho_2(g_i - x_i(t-1))$$

Zusätzlich ist $v_i(t)$ auf ein Intervall $[-V_{\max}, V_{\max}]$ beschränkt, um zu verhindern, dass das System explodiert. Wenn die Werte diesen Bereich verlassen, werden sie auf $-V_{\max}$ bzw. $V_{\max}$ zurückgesetzt.

Aus dem Richtungsvektor wird dann die neue Position berechnet:

$$x_i(t) = x_i(t-1) + v_i(t)$$

Als letztes wird das lokale Optimum sowie das Optimum der Nachbarn von i aktualisiert, $N(i)$ sei die Menge der Nachbarn von i:

$$p_i = \operatorname{argmin}_{y \in \{p_i, x_i\}} f(y)$$

$$g_i = \operatorname{argmin}_{y \in \{p_j \mid j \in N(i)\} \cup \{p_i\}} f(y)$$

Wird die Nachbarschaft als der gesamte Graph definiert, so kann statt den g_i auch ein einziger Wert verwendet werden, der dann jeweils durch den Vergleich mit den lokalen Werten aktualisiert werden kann.

Im Folgenden werden die verschiedenen Parameter bei Partikelschwärmen nochmals kurz besprochen und Hinweise zu ihrer Wahl gegeben.

Je mehr Partikel in einem Partikelschwarm sind, desto besser sind die Ergebnisse. Gleichzeitig erhöht sich allerdings auch die Laufzeit des Verfahrens, so dass hier ein Kompromiss zwischen Laufzeit und Qualität gefunden werden muss. Typische Größen sind zwischen 20 und 60 Partikeln.

Einen großen Einfluss auf den Verlauf des Verfahrens hat die Wahl der Nachbarschaft von Partikeln. Hierbei gibt es zwischen den Extremen der gesamten Population und einer Ringstruktur mit nur zwei Nachbarn pro Partikeln viele Möglichkeiten wie z.B. auch geometrische Formen wie ein Torus oder ein Hyperwürfel. Eine große Nachbarschaft führt zu einem größeren Einfluss lokaler Maxima und zu einer schnelleren Konvergenz, eine kleinere Nachbarschaft führt zu einer langsameren Konvergenz, aber unter Umständen auch zu einer höheren Qualität. Die optimale Wahl hängt dabei von der Beschaffenheit des konkreten Optimierungsproblems ab.

Die beiden Parameter C_1 und C_2 geben ein oberes Limit für den zufällig gewählten Einfluss der besten eigenen und der besten Lösung der Nachbarn vor. Kleinere Werte führen hier zu einer langsameren Bewegung und daher besseren Erforschung einer näheren Umgebung der Partikel, hohe Werte führen zu mehr Bewegung und können auch zu Oszillation führen. Ein typischer Wert für beide Parameter ist zwei.

Das obere Limit für die Geschwindigkeit $V_{\max}$ hängt von der konkreten Problemstellung ab. An Stelle des Limits $V_{\max}$ kann auch die Trägheit w in Abhängigkeit von ρ_1 und ρ_2 gewählt werden und so eine zu hohe Geschwindigkeit der Partikel verhindert werden.

Die verschiedenen Parameter können auch dynamisch bzw. adaptiv gewählt werden. So kann z.B. w von 0,9 am Anfang im Laufe der Zeit auf 0,4 gesenkt werden. Auch die Anzahl der Partikel kann variiert werden, so kann z.B. der Partikel mit der schlechtesten Lösung entfernt werden.

3.2.1 Diskretisierung von Partikelschwärmen

Betrachtet man Partikelschwärme, so ist zunächst unklar, wie das Verfahren für diskrete Probleme verwendet werden kann. Man muss hier zwei Aspekte betrachten: Die Positionen der Partikel und die Bewegungsvektoren der Partikel. Die Positionen der Partikel könnten z.B. als binärer Vektor kodiert werden, die Bewegungen können stochastisch modelliert werden. Eine Möglichkeit für die Modellierung der Bewegung bei binären Vektoren ist, zunächst jeden Eintrag v_d des Richtungsvektors mit einer Sigmoid-Funktion in das Intervall $[0, 1]$ zu projizieren:

$$S(v_d) := \frac{1}{1 + \exp(-v_d)}$$

Anschließend kann mit einer zufällig gewählten Schwelle im selben Intervall entschieden werden, ob die zugehörige (binäre) Position auf 1 bzw. 0 gesetzt wird wenn $S(v_d)$ größer gleich bzw. kleiner dieser Schwelle ist. Je höher die „Geschwindigkeit“ in die entsprechende Richtung ist, desto wahrscheinlicher ist es damit, dass die entsprechende binäre Variable in der Position des Partikels auf 1 gesetzt wird.

Eine andere Modellierung, die von Qin et al. [QYB11] vorgestellt wurde, ist, eine Übergangsfunktion zwischen den diskreten Zuständen zu definieren. Durch Anwendung dieser Funktion bewegt man sich in einem Zustandsgraph, ein Pfad in diesem Graphen entspricht einer Folge von Anwendungen der Übergangsfunktion. Hierdurch kann zum einen der Abstand zwischen zwei Zuständen als die Länge der kürzesten Folge von Anwendungen der Übergangsfunktion (diese entspricht dem kürzesten Pfad im Zustandsgraph) und zum anderen auch der Begriff eines Richtungsvektors als diese Folge von Anwendungen der Übergangsfunktion definiert werden. Die Differenz zwischen Zuständen ist entsprechend der kürzeste Pfad zwischen den beiden Zuständen.

Eine Multiplikation einer reellen Zahl $a \in [0, 1]$ mit einem Richtungsvektor wird als Verkürzung der Folge von Anwendungen der Übergangsfunktion aufgefasst. Außerdem wird das Modell des Partikelschwarms in der Hinsicht vereinfacht, dass immer nur entweder die beste eigene Lösung oder die beste Lösung der Nachbarn berücksichtigt wird, aber nie beide. Die Wahrscheinlichkeit, dass die lokale beste Lösung als Beeinflussungsfaktor verwendet wird, ist $prob_p$, die Wahrscheinlichkeit, dass die beste Lösung der Nachbarschaft verwendet wird, entsprechend $prob_n = 1 - prob_p$. Durch den Vergleich mit einem zufällig aus $[0, 1)$ gezogenen Wert δ mit $prob_p$ wird entschieden, welche der beiden Lösungen verwendet wird.

Der sich daraus ergebende Partikelschwarmoptimierungs-Algorithmus für diskrete Modelle ist in Algorithmus 1 dargestellt.

3.3 Beispiel: Knotenfärbung bei Graphen

Um eine bessere Vorstellung davon zu vermitteln, wie populationsbasierte Metaheuristiken für praktische Probleme verwendet werden können, sollen hier nun zwei populationsbasierte Metaheuristiken für das Knotenfärbungsproblem bei Graphen vorgestellt werden.

Das Ziel bei der Knotenfärbung von Graphen ist, jedem Knoten eine Farbe zuzuweisen, so dass keine zwei adjazente Knoten die selbe Farbe haben. Die minimale Anzahl an Farben, mit der dies bei einem Graphen G möglich ist, wird die *chromatische Zahl* $\chi(G)$ genannt. Das Entscheidungsproblem, ob ein Graph mit k Farben gefärbt werden kann, ist

Algorithm 1: Diskreter Partikelschwarmoptimierungs-Algorithmus

Input: Ein diskretes Optimierungsproblem
Output: Eine möglichst optimale Lösung

$S \leftarrow N$-dimensionaler Schwarm;
Initialisiere S;
repeat
  foreach $i \in S$ **do**
    if $f(x_i) < f(p_i)$ **then**
      $p_i \leftarrow x_i$;
  foreach $i \in S$ **do**
    $g_i \leftarrow \text{argmin}_{p_j}\{f(p_j) \mid j \in N(i) \subset S\}$;
  foreach $i \in S$ **do**
    Wähle $\delta \in [0, 1)$ zufällig;
    if $0 \leq \delta < prob_p$ **then**
      $v_i \leftarrow r_1(p_i - x_i)$;
    else
      $v_i \leftarrow r_2(g_i - x_i)$;
    $x_i \leftarrow x_i + v_i$;
until *Abbruchbedingung erfüllt*;

NP-vollständig. Das Knotenfärbungsproblem ist derartig schwer, dass die chromatische Zahl bereits für bestimmte Graphen mit wenigen hunderten Knoten unbekannt ist.

Sei $G = (V, E)$ ein ungerichteter *Graph* mit Knotenmenge V und Kantenmenge $E \subseteq V \times V$. Eine *Färbung* ist durch eine Abbildung $c : V \to \{1, \ldots, k\}$ gegeben. Eine Färbung heißt *gültig*, wenn $\forall (u, v) \in E : c(u) \neq c(v)$.

Es ist leicht zu sehen, dass $\chi(G) \leq |V|$ ist. Algorithmen, die lediglich bei gegebenem k eine k-Färbung zu berechnen versuchen, können somit auch leicht in Algorithmen zur Berechnung von $\chi(G)$ umgewandelt werden. Dies bedeutet, dass sich das Entscheidungsproblem, ob ein Graph mit k Farben gefärbt werden kann, und die Berechnung von $\chi(G)$ von der Laufzeit her maximal um einen linearen Faktor unterscheiden.

Typische Lösungsansätze für das Knotenfärbungsproblem basieren auf Heuristiken, zum einen gibt es problemspezifische Heuristiken, zum anderen wurden Verfahren unter Verwendung von Metaheuristiken entwickelt, es gibt unter anderem Ansätze basierend auf genetischen Algorithmen, Ameisenalgorithmen und Partikelschwärmen.

Im Folgenden werden zum einen ein Ansatz von Dowsland und Thompson [DT08] mit einem Ameisenalgorithmus und zum anderen ein Ansatz von Qin et al. [QYB11] mit einem Partikelschwarm vorgestellt.

Eine alternative Auffassung des Knotenfärbungsproblems ist, dass eine möglichst kleine Menge von disjunkten unabhängigen Knotenmengen konstruiert werden soll, d.h. Mengen von Knoten, so dass in einer Menge keine zwei Knoten benachbart sind. Durch diese alternative Formulierung werden Färbungen, die sich lediglich durch Umbenennung von Farben unterscheiden und damit äquivalent sind, identisch.

3.3.1 Ameisenalgorithmus

Der Algorithmus konstruiert in jedem Iterationsschritt eine Reihe von Färbungen, die Färbungen an sich werden dabei jedes Mal neu aufgebaut. Zum Aufbau der Färbungen wird ein Greedy-Verfahren verwendet, das eine Farbklasse nach der anderen aufbaut. Hierbei werden zwei verschiedene Heuristiken verwendet, zum einen wird der Ameisenschwarm dazu verwendet, Knoten, die in vorher berechneten Färbungen mit der selben Farbe gefärbt wurden, wahrscheinlicher erneut mit der selben Farbe zu färben, wenn sie Teil einer guten Lösung waren, zum anderen gibt es verschiedene Heuristiken um zu bewerten, ob ein Knoten in eine bestimmte Farbklasse aufgenommen werden sollte.

Um zu entscheiden, ob ein bestimmter Knoten in eine Farbklasse aufgenommen werden soll, werden eine Reihe von Variablen definiert: Die Menge W sei die Menge der noch nicht gefärbten Knoten, die in die aktuelle Farbklasse k noch aufgenommen werden könnten, sprich jeweils keine Nachbarn in der Menge C_k, der Menge der Knoten in der Farbklasse k haben. Umgekehrt sei B die Menge der noch nicht gefärbten Knoten, die nicht in die aktuelle Farbklasse aufgenommen werden können, also Nachbarn in C_k haben. Außerdem sei $deg_X(i)$ der Grad des Knotens i im Subgraph von G, der von der Knotenmenge X

induziert wird.

Die Variable η_{ik} sei das Kriterium, ob ein Knoten i in der Farbe k gefärbt werden soll. Es gibt nun drei Möglichkeiten, die von Dowsland und Thompson vorgestellt werden, η_{ik} zu definieren, hier jeweils mit einer kurzen Überlegung, weshalb diese Heuristik Sinn machen könnte:

- $\eta_{ik} := \deg_B(i)$ – Knoten, die viele Nachbarn haben, die in der aktuellen Farbe nicht gefärbt werden können, sollten aufgenommen werden, da sie nicht zusammen mit den anderen Knoten in B in einer Farbe gefärbt werden können
- $\eta_{ik} := |W| - \deg_W(i)$ – Knoten, die einen geringen Grad in W haben, verhindern bei möglichst wenig Knoten in W, dass sie anschließend ebenfalls hinzugefügt werden können
- $\eta_{ik} := \deg_{B \cup W}(i)$ – Knoten, die viele Nachbarn haben, die noch nicht gefärbt wurden, sollten mit der aktuellen Farbe gefärbt werden, da es sonst nur noch wenige Möglichkeiten gibt, sie zu färben.

Für die Wahl des ersten Knotens einer Farbklasse hat sich bei früheren Experimenten herausgestellt, dass eine zufällige Wahl die beste Option ist.

Die Pheromonmatrix wird für diesen Algorithmus über die Knoten indiziert, am Anfang werden alle Einträge von adjazenten Knotenpaaren auf 0 gesetzt, alle von nicht-adjazenten Knoten auf 1. Nach jeder Runde werden die Einträge t_{ij} durch $\rho t_{ij} + \sum_{s \in S_{ij}} 1/q(s)$ ersetzt. Hierbei ist S_{ij} als Menge der Lösungen definiert, in denen i und j in der selben Farbklasse sind und $q(s)$ gibt die Menge der benötigten Farben in Lösung s an. Der Parameter ρ gibt an, wie stark die Pheromone der letzten Runde verringert werden sollen.

Der Parameter τ_{ik}, der basierend auf der Pheromonmatrix angibt, wie gut es wäre, den Knoten i mit Farbe k zu färben, ist durch folgende Formel definiert:

$$\tau_{ik} := \frac{\sum_{j \in V_k} t_{ij}}{|V_k|}$$

Um diese Faktoren τ_{ik} und η_{ik} in eine Wahrscheinlichkeit P_{ik} zu überführen, mit der ein Knoten i mit der Farbe k gefärbt wird, werden die beiden Faktoren mit Exponenten α bzw. β gewichtet und normalisiert:

$$P_{ik} := \frac{\tau_{ik}^{\alpha} \eta_{ik}^{\beta}}{\sum_{j \in W} \tau_{jk}^{\alpha} \eta_{jk}^{\beta}} \text{ falls } i \in W, \text{ sonst } P_{ik} := 0$$

Insgesamt ergibt sich damit Algorithmus 2.

Dieser Algorithmus wurde dann von Dowsland und Thompson noch weiter verbessert, bis er entweder gleich gute oder nur leicht schlechtere Ergebnisse wie aktuelle Algorithmen für das Knotenfärbungsproblem lieferte. Hier wird lediglich kurz auf die jeweiligen Punkte

Algorithm 2: Ameisenalgorithmus für das Knotenfärbungsproblem

$t_{ij} \leftarrow 1 \quad \forall\, i, j \in V,\ i \neq j;$
foreach *Iteration* **do**
  $\delta_{ij} \leftarrow 0 \quad \forall\, i, j \in V,\ i \neq j;$
  foreach *Ameise* **do**
    $X \leftarrow V;$
    $k \leftarrow 0;$
    while $X \neq \emptyset$ **do**
      $k \leftarrow k + 1;$
      $C_k \leftarrow \emptyset;$
      $W \leftarrow X;$
      Wähle $i \in W$ gleichverteilt zufällig;
      Färbe i mit Farbe k;
      while $W \neq \emptyset$ **do**
        Wähle $j \in W$ mit Wahrscheinlichkeit P_{jk};
        Färbe j mit Farbe k;
    $\delta_{ij} \leftarrow \delta_{ij} + 1/k \quad \forall i, j\colon\ C_i = C_j,\ i \neq j;$
  $t_{ij} \leftarrow \rho t_{ij} + \delta_{ij} \quad \forall i, j \in V,\ i \neq j;$
Färbe i mit Farbe k:
$X \leftarrow X \setminus \{i\};$
$C_k \leftarrow C_k \cup \{i\};$
$W \leftarrow W \setminus (N(i) \cup \{i\});$

eingegangen werden, Details können in dem Papier von Dowsland und Thompson [DT08] nachgelesen werden.

Dowsland und Thompson haben den Algorithmus modifiziert, so dass er aufhört, wenn eine bestimmte Anzahl Farben erreicht wurde. Um auszugleichen, dass die Wahrscheinlichkeiten für Knoten, die keiner Farbklasse angehören, nicht erhöht werden, wurde ein zusätzlicher Faktor für P_{ik} eingeführt, der erhöht wird, wenn ein Knoten in einer Runde keine Farbe bekommen hat.

Außerdem wurde mit der Bewertungsfunktion experimentiert, zum einen mit der Anzahl der nicht gefärbten Knoten (statt einfach der Anzahl Farben), zum anderen wurden Knoten, die nicht mehr gefärbt werden konnten, zu der Farbklasse hinzugefügt, bei der sie am wenigsten Konflikte erzeugen und die Anzahl der Konflikte wurde als Bewertung der Lösung verwendet, um feiner zwischen verschiedenen Lösungen differenzieren zu können.

Ein wesentlicher Verbesserungsfaktor stellte die Verwendung lokaler Optimierung dar, zum einen kann man die Farben umsortieren und dann versuchen, Knoten aus höheren Farben in niedrigere Farben zu verschieben und damit eventuell neue Knoten färben zu können. Zum anderern kann man mit einer Art Tiefensuche versuchen, einen Knoten, der aktuell nicht gefärbt werden kann, in die Farbklasse einzufügen, in der er die wenigsten Konflikte hat und dann rekursiv die Knoten, die dann Konflikte haben, wiederum in andere Farbklassen einsortieren usw.. Um Zyklen zu vermeiden, wird dabei jeder Knoten nur einmal betrachtet. Statt systematisch mit einem Baum zu suchen kann diese Suche auch mit Tabusuche durchgeführt werden, hierbei werden Knoten zwischen den Farbklassen verschoben, wenn das die Bewertungsfunktion verbessert, eine Klasse ist dann für eine gewisse Anzahl Runden für diesen Knoten „tabu“.

Insgesamt wurde so eine dramatische Verbesserung der Qualität der Ergebnisse der Knotenfärbung mit Ameisenalgorithmen gegenüber einer früheren Version des Algorithmus ohne die genannten Verbesserungen erreicht.

3.3.2 Partikelschwarmoptimierung

Für die Lösung des Knotenfärbungsproblems mit Partikelschwarmoptimierung wird das Problem der k-Färbbarkeit betrachtet, sprich ob es eine Färbung mit k Farben gibt bzw. die Berechnung dieser Färbung. Hierbei wird nicht in jeder Iteration eine komplett neue Färbung generiert, sondern zunächst eine Menge initialer Färbungen erzeugt, die dann iterativ verbessert werden.

Als einen Zustand eines Partikels betrachtet man eine Färbung, sprich eine Zuordnung der Knoten in disjunkte Mengen, als Operator wird das Verschieben von Knoten zwischen diesen Mengen definiert. Die Distanz zwischen zwei Färbungen sei die kleinste mögliche Anzahl an Verschiebungen von Knoten zwischen diesen Mengen bzw. Partitionen. Diese kleinste mögliche Anzahl an Verschiebungen kann in $O(|V|^3)$ Zeitkomplexität berechnet werden.

Wie auch schon bei dem Ameisenalgorithmus wird auch von Qin et al. Tabusuche verwendet, um die Lösungen lokal zu verbessern.

Um die Suche nach einer Knotenfärbung weiter zu diversifizieren wird zusätzlich zu der besten lokalen Lösung sowie der besten Lösung der Nachbarschaft noch eine zufällig generierte Färbung mit einer gewissen Wahrscheinlichkeit berücksichtigt, der Unterschied zwischen der aktuellen Lösung und der zufälligen Färbung wird genauso wie die anderen beiden Unterschiede mit einem zufälligen Faktor multipliziert.

Da die initiale Lösung bei einer zufälligen, gleichverteilten Zuteilung einzelner Knoten zu den jeweiligen Farben die Farben jeweils ungefähr gleich große Knotenmengen hätten, werden bei der Initialisierung zunächst zufällige Zahlen n_i, $i = 1, 2, \ldots, k$ mit $\sum_{i=1}^{k} n_i = |V|$ generiert. Anschließend werden dann den jeweiligen Farben $j = 1, \ldots, k$ jeweils n_i Knoten zufällig zugeordnet.

Auf acht Testinstanzen, die aus der zweiten DIMACS Implementation Challenge stammen, wurde sowohl der Ameisenalgorithmus von Dowsland und Thompson als auch der Partikelschwarmalgorithmus getestet. Beide Algorithmen konnten auf den getesteten Graphen gute Ergebnisse erreichen. Bei manchen Graphen erreichten beide Algorithmen nicht das mögliche k, bei anderen Instanzen nur jeweils einer der beiden.

4 Fazit

Populationsbasierte Metaheuristiken zeigen, dass in der Natur beobachtete Phänomene gut in die Informatik übertragen und zur Approximation der Lösung schwerer Probleme verwendet werden können. Durch eine geeignete Modellierung können auch Probleme, bei denen man sich zunächst weder einen Zusammenhang mit Ameisenkolonien noch mit Vogelschwärmen vorstellen kann, auf entsprechende Verfahren abgebildet werden.

Entscheidend ist hierbei, dass das Problem geeignet modelliert wird, die Zielfunktion aussagekräftig ist und ein geeignetes Mittel zwischen einer großen Diversität an Lösungen und einer doch für die Praxis ausreichend schnelle Konvergenz gefunden wird.

Die einzelnen Verfahren bieten bereits an sich eine große Anzahl an Parametern, kombiniert mit problemspezifischen Parametern kann dies schnell sehr komplex werden, so dass viele Tests benötigt werden, um geeignete Werte zu finden.

Wie man am Beispiel des Knotenfärbungsproblems bei Graphen gesehen hat, sind Metaheuristiken oft aber auch nur eine Komponente eines deutlich umfangreicheren Frameworks an Optimierungsverfahren und sind an sich noch nicht in der Lage, wirklich gute Lösungen zu finden. Erst durch eine geschickte Wahl der Problemstellung und durch die Kombination mit weiteren Heuristiken zeigen populationsbasierte Metaheuristiken ihre Stärken.

Insgesamt betrachtet sind populationsbasierte Metaheuristiken eine Komponente, die, richtig eingesetzt, beachtlich gute Lösungen produzieren kann.

Literaturverzeichnis

[DT08] Dowsland, Kathryn A. ; Thompson, Jonathan M.: An improved ant colony optimisation heuristic for graph colouring. In: *Discrete Appl. Math.* 156 (2008), Februar, Nr. 3, 313–324. http://dx.doi.org/10.1016/j.dam.2007.03.025. – DOI 10.1016/j.dam.2007.03.025. – ISSN 0166–218X

[MMP07] Maaranen, Heikki ; Miettinen, Kaisa ; Penttinen, Antti: On initial populations of a genetic algorithm for continuous optimization problems. In: *Journal of Global Optimization* 37 (2007), 405-436. http://dx.doi.org/10.1007/s10898-006-9056-6. – ISSN 0925–5001. – 10.1007/s10898-006-9056-6

[QYB11] Qin, Jin ; Yin, Yixin ; Ban, Xiaojuan: Hybrid Discrete Particle Swarm Algorithm for Graph Coloring Problem. In: *JCP* 6 (2011), Nr. 6, S. 1175–1182

[Tal09] Talbi, E.G.: *Metaheuristics: From Design to Implementation*. John Wiley & Sons, 2009 (Wiley Series on Parallel and Distributed Computing). http://books.google.de/books?id=SIsa6zi5XV8C. – ISBN 9780470278581